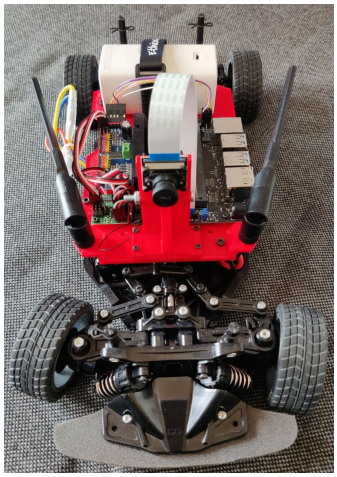
Objektverfolgung auf Eingebetteten Systemen

Albert Sauer
Department Informatik, Hochschule für angewandte Wissenschaften Hamburg
albert.sauer@haw-hamburg.de



Zusammenfassung

Auf einem Jetson Nano wird ein Hausschuh mithilfe von einem Faltungsnetz erkannt und verfolgt. Die Verfolgung des Objekts wird dabei durch ein Ferngesteuertes Auto im Maßstab von 1:10 durchgeführt, welches von dem Jetson Nano über PWM-Signale gesteuert wird. Die Objekterkennung geschieht in Echtzeit, dabei wird das Objekt mit durchschnittlich 31.8 Bildern pro Sekunde erkannt. Um dies zu erreichen wurde das trainierte Modell mithilfe von TensorRT quantisiert. Des Weiteren wurde gezeigt, dass das Faltungsnetz durch den Einsatz von Daten Augmentation weiter verbessert werden kann, um so eine höhere Reichweite und Genauigkeit der Objekterkennung zu erreichen.

Keywords: Jetson Nano, Objekterkennung, Objektverfolgung, Echtzeit, Faltungsnetz, TensorRT, Faltungsnetz, Pulsdauermodulation, Daten Augmentation

1 Einleitung

Um einem Objekt folgen zu können benötigt man eine Hardware die kompakt, beweglich und leistungsstark ist. Um dies zu erreichen, wird die Kombination aus einem TamiyaCar[1] und einem Jetson Nano[2] verwendet. Um ein gesuchtes Objekt im Bild zu finden wurde eine Objekterkennung statt einer Objektklassifizierung gewählt. Da man mit der Objekterkennung das Objekt im Bild lokalisieren kann, werden diese Informationen für die Objektverfolgung benötigt.
Als Architektur wurde ein SSD[3] gewählt, dabei handelt es sich um das „Mobilenet v2“ [4]. Dieses neuronale Netz bietet eine geringe Inferenzzeit bei mäßiger Genauigkeit [5]. Dies wird benötigt, um in Echtzeit auf die Umwelt reagieren zu können und somit ohne Verzögerung Objekte verfolgen zu können. Des Weiterenmuss für die Verfolgung eine Ablaufsteuerung entwickelt werden, bei der eine Abstandsregel eingehalten wird, sowie entschieden wird, wann das Auto lenken muss. Ohne diese Ablaufsteuerung kann es zu Problemen kommen bei dem das Objekt angefahren wird.

2 Hardware und Materialien

2.1 Komponenten

In diesem Abschnitt wird erläutert welche Hardware für die Umsetzung der Ausarbeitung benutzt wurde und auf welchen bereits vorhandenen Projekten diese Arbeit aufbaut. Als Inspiration wurden vor allem Beispiel Projekte gewählt, die vom Hardware-Hersteller Nvidia angeboten wurden.[6] [7]

Verwendete Hardware:

- Jetson Nano Devoloper Kit 4GB RAM
- Ferngesteuertes Modellauto 1:10 (Tamiya)
- 7,4V NiMh-Akku
- PCA9685 12 Kanal PWM Servo Treiber
- 5V 2A Powerbank
- Pololu 4-Kanal RC Servo Multiplexer
- IMX219-160° Kamera
- 32GB Speicher SD-Karte
- Wifi-Netzwerkkarte
- WLAN Antenne
- Verbindungskabel
- 3D-Druck Bauteile

2.2 Aufbau des Jetson Nano Auto

In diesem Abschnitt wird erläutert welche Funktionsweise die einzelnen Komponenten haben und wie diese miteinander kommunizieren.

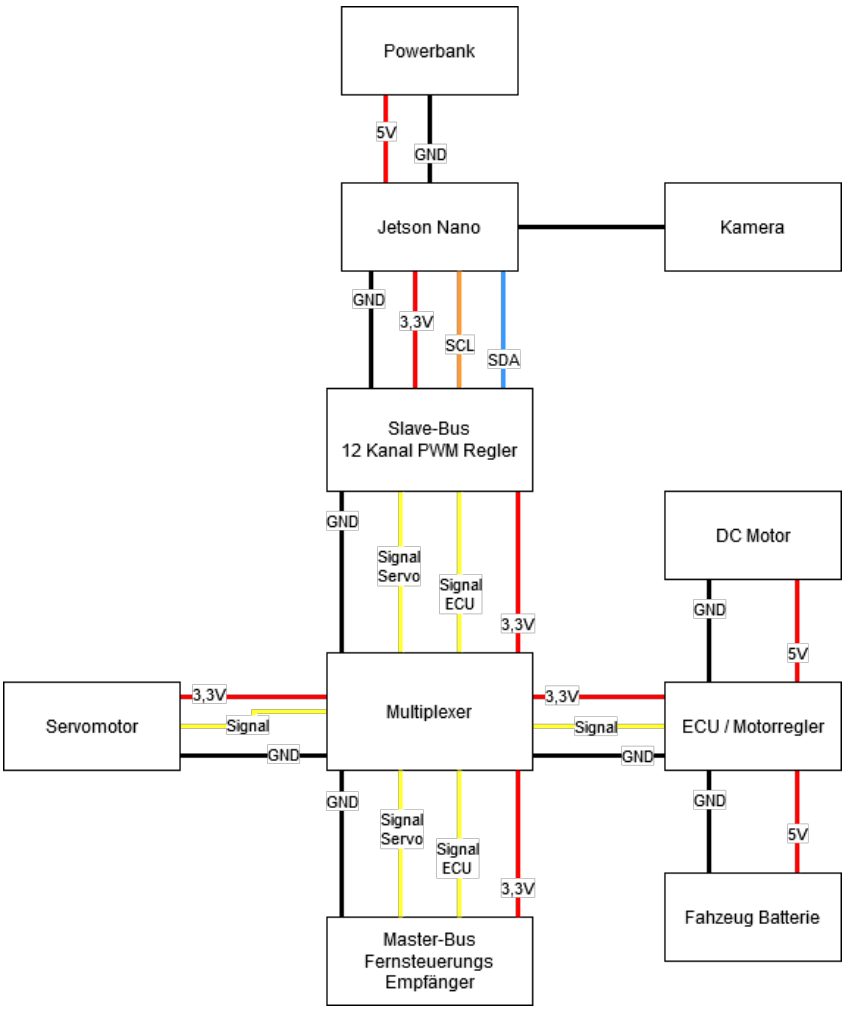


Abbildung 1: Schaltplan für den Hardware Aufbau

Auf Abbildung 1 sieht man, dass der Jetson Nano über eine Powerbank mit Strom versorgt wird. Zudem ist an diesem die Kamera über ein Flachbandkabel angeschlossen was hier nur als Strich visualisiert wird. Damit der Jetson Nano befehle für die Steuerung ausführen kann, kommuniziert dieser über I^2C (SCL, SDA) mit dem PWM Regler. Der PWM-Regler schickt dabei ein PWM-Signal an den Multiplexer, welcher dieses Signal an die jeweilige adressierte Komponente weiterleitet.
Der Multiplexer dient außerdem dazu, zwischen einer manuellen Fernsteuerung und der Steuerung vom Jetson Nano umzuschalten. Dadurch wird ermöglicht, dass ein Benutzer bei einem Fehlverhalten des Autos, jederzeit eingreifen kann.

3 Steuerungssoftware

Um einem Objekt folgen zu können benötigt man eine Software, die anhand der erkannten Objekte den Lenkwinkel und die Fahrtrichtung kontrolliert. Dazu wurde eine Statemachine entwickelt.

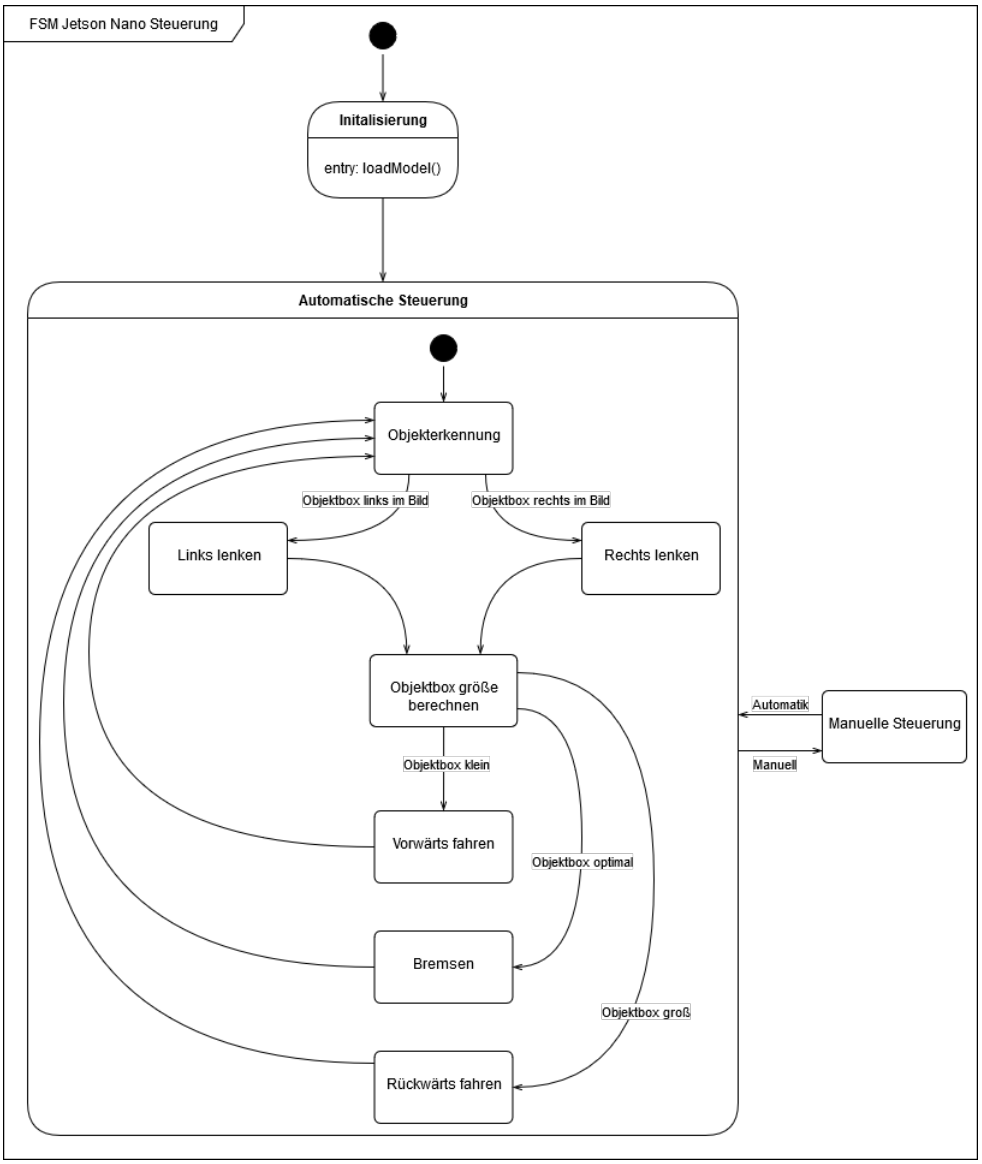


Abbildung 2: Statemachine für Ablaufsteuerung

Auf der Abbildung 2 sieht man nun wie sich das Auto bei den Eingaben verhalten soll. Dabei wird differenziert, ob das Auto grade manuell oder automatisch gesteuert wird. Bei der automatischen Steuerung wird zunächst der Lenkwinkel anhand der Objektbox bestimmt, die von der Objekterkennung zurückgegeben wird. Danach kann mittels der Objektboxgröße die Distanz zum Objekt bestimmt werden und entschieden werden in welche Richtung das Auto fahren soll.

4 Objekterkennung

4.1 Modell Auswahl

Für die Objekterkennung wurde ein SSD gewählt wie es in Abbildung 3 zu sehen ist. Dabei wird ein Modell genommen, welches eine geringe Inferenzzeit hat, um so möglichst Echtzeitfähig zu sein.
Dafür eignet sich das vortrainierte „Mobilenet v2“ am besten für diesen Anwendungsfall. Dieses Modell besitzt eine mAP von 20.2 bei einer Inferenzzeit von 19 ms[4]. Für die Objekterkennung werden die erhaltenen Bilder von der Kamera auf 300x300x3 Pixel dimensioniert. Dies ist die einzige Daten Vorverarbeitung dir für die Objekterkennung angewandt wird.

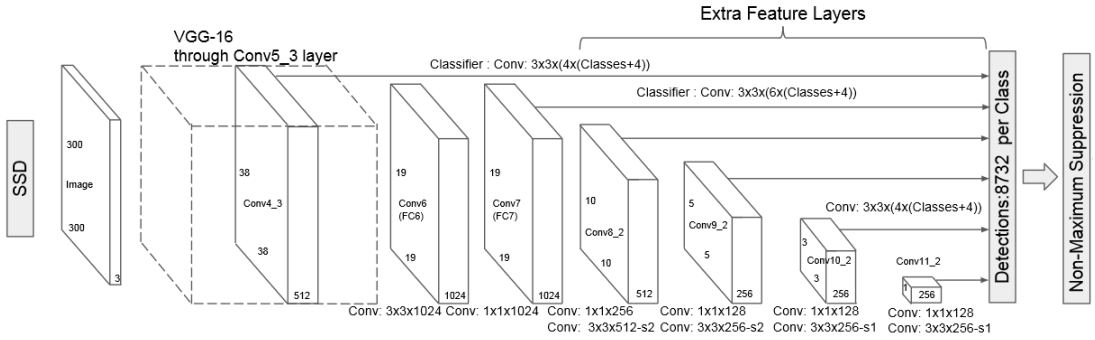


Abbildung 3: Aufbau eines Single Shot MultiBox Detectors[3]

4.2 Training des Neuronalen Netzes

Das vortrainierte Modell soll mithilfe einer Objekterkennung API darauf trainiert werden einen bestimmten Hausschuh im Bild zu erkennen. Dafür wurden 364 Bilder von diesem Hausschuh erstellt und anschließend gelabelt. Aus diesem Datensatz werden 328 Bilder für das Training benutzt und 36 Bilder um anschließend das trainierte Modell zu testen. Dabei wurden viele Bilder mit verschiedenen Hintergründen generiert, um so eine höhere Varianz zwischen den Bildern zu haben. Außerdem wurde die Entfernung des Objekts zur Kamera verändert, dies soll es ermöglichen viele verschiedene Objektboxen für die Distanzberechnung zu generieren.

Abbildung 4: Bilder des Training Datensatz vom Objekt

In dieser Abbildung ist nun das Objekt zu sehen, auf welches das Modell für die Objekterkennung trainiert wurde. Darauf erkennt man nun auch die Varianz der unterschiedlichsten Hintergründe.

4.3 Erweiterung der Datensatzes

Um die Objekterkennung robuster zu machen wurde Daten Augmentation[8] eingesetzt, dabei wird der Datensatz künstlich erweitert um so mehr Bilder aus den bereits vorhandenen Bildern zu generieren. Dies ermöglichte einen Datensatz mit 874 Bildern zu generieren. Dabei wurden 787 für das Training verwendet und 87 für das Testen. Bei den Test Bildern allerdings nur die Original Bilder verwendet. In den Ergebnissen wird anschließend verglichen, wie sich dieses Verfahren auf die Genauigkeit ausgewirkt hat.

Abbildung 5: Aus dem Datensatz generierte Bilder

Auf der Abbildung 5 sieht man nun die mithilfe eines Tools[9] generierten Bilder. Um die Bilder zu generieren wurden verschiedene Effekte angewandt. Bei diesen Effekten geht es um Drehung, Zerrung, Helligkeit und Kontrast verändern und hinzufügen von Rauschen. Alle Effekte wurden nur mit einem geringen Anteil hinzugefügt.

5 Evaluierung der Ergebnisse

Um die Objekterkennungsmodelle zu vergleichen wird hier eine „mean Average Precision“ verwendet[10]. Bei diesem Verfahren wird mithilfe der berechneten Objektboxen die Genauigkeit des Modells berechnet. Dabei werden die richtig erkannten Pixel der Objektbox (TP) mit den falsch erkannten Pixeln (FP) verrechnet und davon der Durchschnitt über die Anzahl der Testbilder genommen.

$$mAP = \frac{1}{n} \sum_{k=1}^{n} \frac{TP_k}{TP_k + FP_k}$$

5.1 Auswirkung von Daten Augmentation auf das Training

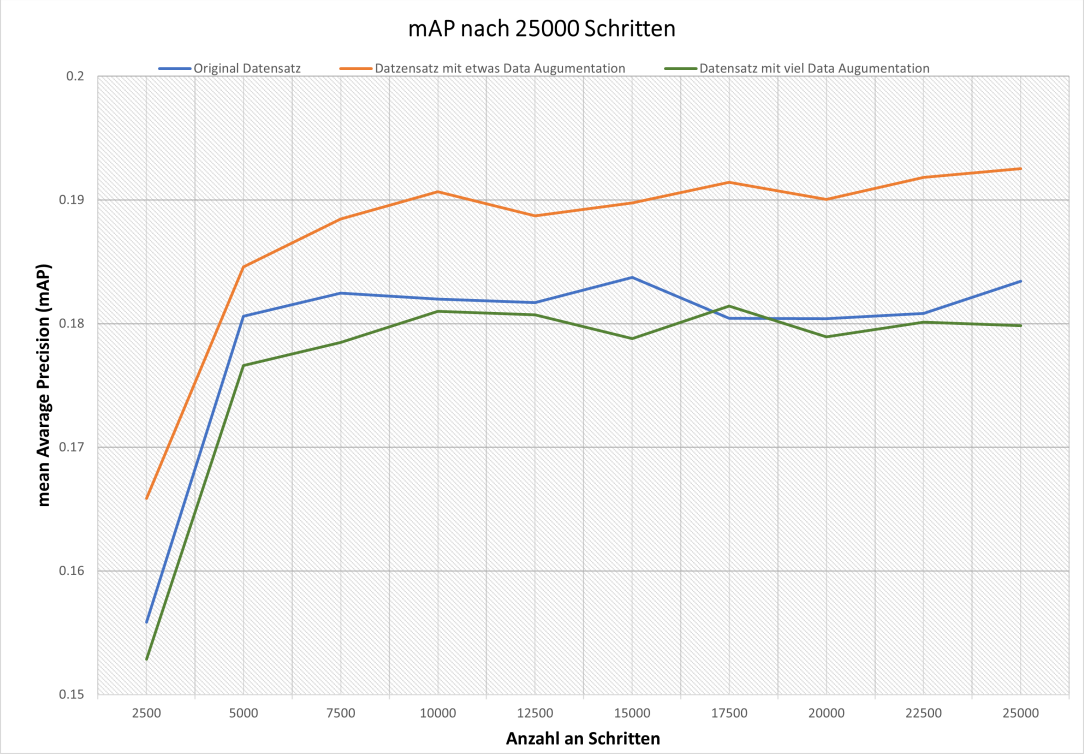


Abbildung 6: Genauigkeit der Objekterkennungsmodelle

Auf Abbildung 6 ist zu sehen, wie sich die Genauigkeit der Objekterkennung durch geringes hinzufügen von Daten Augmentation verbessert hat. Außerdem erkennt man, dass bei zu viel Augmentation sich dies negativ auf die Genauigkeit auswirkt.

5.2 Auswirkung von Daten Augmentation auf die Objektverfolgung

Die Daten Augmentation hatte zu dem Auswirkungen auf die Distanz bei der, der Hausschuh erkannt wurde. Hierfür wurde ein Vergleich zwischen dem Original Datensatz und dem Datensatz mit etwas Daten Augmentation erstellt. Für diesen Vergleich wurde ein Threshold von 0.5 gewählt, um so die maximale Distanz bei minimaler Fehlerkennung zu erlangen.

Entfernung	**Modell Genauigkeit in Prozent**	
	ohne Augmentation	**mit Augmentation**
20 cm	99,7%	99,8%
50 cm	93,3%	96,1%
1 m	78,2%	85,0%
1.5 mm	61,0%	73,4%
2 m	keine Objekterkennung mehr möglich	53%

Tabelle 1: Messung zur Reichweite der Objekterkennung

5.3 Echtzeit Objekterkennung auf dem Jetson Nano

Um die Echtzeitfähigkeit der Objekterkennung auf dem Jetson Nano zu messen wurden die Bilder pro Sekunde (FPS) berechnet und anschließend mit dem originalen Mobilenet v2 und mit dem eigens trainierten Mobilenet v2 verglichen. Da das originale Mobilenet v2 90 Klassen besitzt wird angenommen, dass das eigens trainierte Modell mit nur einer Klasse, eine höhere FPS aufweisen wird.

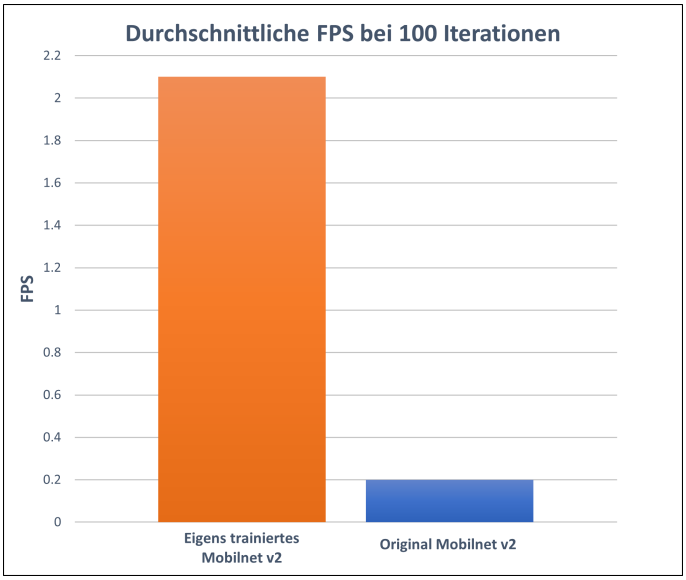


Abbildung 7: Durchschnittliche berechnete FPS der Modelle für die Objekterkennung

Auf Abbildung 7 ist nun deutlich zu erkennen, dass die erreichten 2.1 FPS zu gering für eine Objektverfolgung in Echtzeit ist. Damit die Objekterkennung echtzeitfähig wird, wird das Modell mithilfe von Nvidias TensorRT[11] quantisiert. Dabei werden die Parameter für die Berechnung der Objekterkennung von „Float16“ auf „INT8“ konvertiert. Dies ermöglicht eine drastische Verkürzung der Inferenzzeit und so eine Erhöhung der FPS.

Modell	**Inferenzzeit**	
	ohne TensorRT	**mit TensorRT**
Original Mobilenet v2	2974,2 ms	24,8 ms
Eigens trainiertes Mobilenet v2	1295,3 ms	11,7 ms

Tabelle 2: Inferenzzeit mit und ohne TensorRT

Anhand der Tabelle 2 sieht man nun wie sehr sich die Quantisierung auf die Inferenzzeit auswirkt. Dabei fällt auf, dass die gemessene Inferenzzeit mit TensorRT sich dabei sehr stark, der von den Mobilnet v2[5] vorhandenen Daten ähnelt.

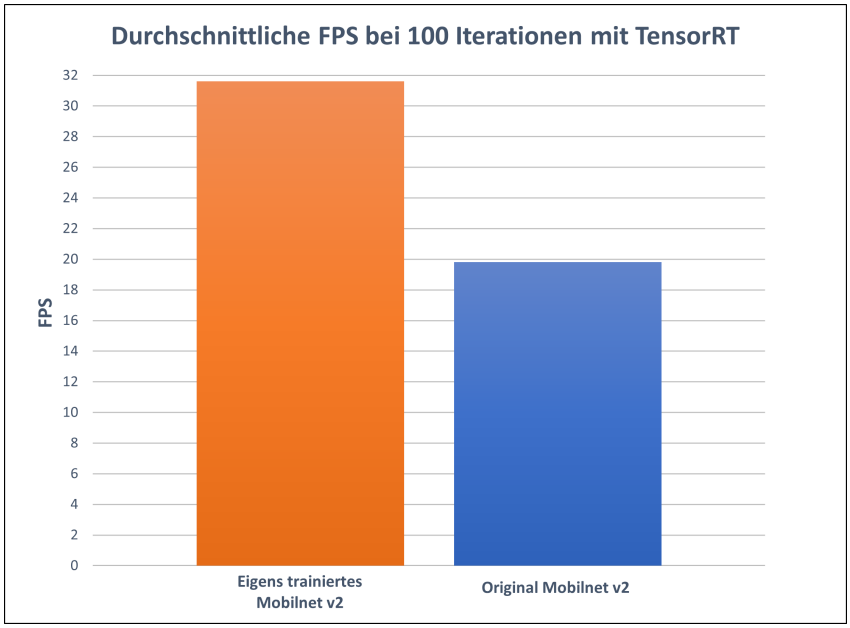


Abbildung 8: Durchschnittliche berechnete FPS der Modelle für die Objekterkennung nach Verwendung von TensorRT

Wie man auf Abbildung 8 erkennen kann, ermöglicht das Quantisieren der Modelle mithilfe von TensorRT eine höhere FPS. Diese Erhöhung macht es möglich

sowohl das Original Modell als auch das eigens trainierte Modell für eine Echtzeitanwendung einzusetzen. Anhand dieser Daten erkennt man auch, dass für ein eigens trainiertes Modell mit wenig Klassen, sogar Modelle mit einer längeren Inferenzzeit und einer höheren mAP gewählt werden können. Als Beispiel dafür würde das „SSD ResNet50“[5] mit einer Inferenzzeit von 87ms und einer mAP von 38.3 die Objekterkennung um einiges robuster machen.

6 Fazit

Diese Arbeit beschäftigte sich damit mithilfe von maschinellem lernen, ein Objekt in Echtzeit auf einem eingebetteten System zu erkennen und zu verfolgen. Es wurden verschiedene Methoden angewandt, um die Objektverfolgung für die Echtzeitanwendung zu verbessern. Dabei wurde bewiesen, dass ein Datensatz mit einer geringen Anzahl an Daten mithilfe von Data Augmentation bereits um ein Vielfaches erweitert werden kann um so Arbeit und Zeit zu sparen. Zudem wurde eine Steuerung zu Objektverfolgung entwickelt, die in Echtzeit durchgeführt werden konnte. Des Weiterenwurde analysiert, wie sich eine Quantisierung der Objekterkennungsmodelle auf die Echtzeitfähigkeit auf dem eingebetteten System auswirkt.
Anhand der Ergebnisse wurde gezeigt, dass es möglich ist eine Objektverfolgung in Echtzeit durchzuführen. Bereits mit geringen Entwicklungs und Hardware-kosten lässt sich so eine Anwendung erweitern, um einfache mobile Assistenzroboter zu entwickeln. Diese können zum Beispiel für bewegungseingeschränkte Menschen als eine Art Transportroboter dienen.

Literatur

[1] URL: https://www.tamiya.de/de/produkte/rc-modelle/.

[2] URL: https://elinux.org/Jetson_Nano.

[3] URL: https://arxiv.org/pdf/1512.02325.pdf.

[4] URL: https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html.

[5] URL: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md.

[6] URL: https://github.com/NVIDIA-AI-IOT/jetracer.

[7] URL: https://github.com/dusty-nv/jetson-inference.

[8] URL: https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/.

[9] URL: https://roboflow.com/.

[10] URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2.

[11] URL: https://developer.nvidia.com/tensorrt.